Software & Systems Modeling (2019) 18:1571–1573
https://doi.org/10.1007/s10270-019-00734-z

EDITORIAL

Conceptual distance of models and languages

Jeff Gray[1] · **Bernhard Rumpe**[2]

Published online: 7 May 2019

Conceptual distance (i.e., measurement of the distance between two sets of concepts) had its roots in linguistics as the semantic distance problem. In the linguistics context, conceptual distance provides a metric of the difficulty in understanding a topic across different disciplines or subject areas. However, it seems that there is no commonly agreed and well-founded theory that allows measurement of the distance between two sets of concepts. We believe that it is worthwhile to measure the conceptual distance between models, and also between languages.

In the domain of conceptual modeling, which emerged from entity–relationship modeling, models are usually explicit, describing the concepts (e.g., classes in OO), their properties (e.g., attributes in OO) and relationships (e.g., associations in OO). The conceptual distance between two models can be measured syntactically by comparing commonly occurring concepts, properties and relationships, versus elements that only exist in one of the models. This is easy to implement, but does not reflect that semantically equivalent models may exist, even if their syntactic representations differ. The syntactic differences may emerge from the use of synonyms for similar terms, or from structurally equivalent but inequal concept/relationship formalizations. A conceptual distance measurement based on semantic differences should consider the semantics of two conceptual models and compare the sets of described instances in a meaningful way. In conceptual modeling, semantics is usually an infinite set of instances (e.g., object structures in OO). Conceptual distance may be difficult to measure accurately.

In a development project, it is often desirable to identify differences of evolved versions of models. This is true for all modeling languages such as code, state machines, class diagrams, use case diagrams or any specific DSL. A pure syntactic interpretation of the conceptual distance can be defined generically by comparing the two models of a language and counting differences and similarities for each language construct (e.g., comparing "classes," "attributes," "states" or any other language construct). A somewhat smarter syntactic measurement could count "compatible" types as similar elements (e.g., int and long). Other similarity metrics could be based on the renaming of an element using a synonym. Such smart measurements exist and can assist in identifying variants of models in a product line development setting.

If the meaning of a model (i.e., its semantics) shall be used to measure conceptual distance of two models, then the situation becomes rather diverse. For the category of structural models (e.g., conceptual models, architectural models, block or class diagrams), conceptual distance of two models needs to compare the sets of possible instantiation structures (which are sets of sets of elements, e.g., objects). For the category of behavioral models, the set of possible behavior executions (e.g., sets of traces) and their differences should be analyzed. The conceptual distance of behavioral descriptions can be defined inductively over time, because there is a starting point, namely time point 0, from where all behaviors initiate. Similarity of behavior descriptions should typically regard the time point of the first deviation of two behavioral executions: The earlier the deviation can be observed, the larger the distance. All of these considerations are complicated by non-deterministic behavior of the described implementation, non-deterministic behavior of the environment, under specification of the models (which is not exactly the same, but similar to a non-deterministic implementation) and the possibility to add probabilities to all of the relevant concepts. This also contributes to the challenge of measuring conceptual distance accurately.

Beyond comparing individual model instances, it is also possible to compare the conceptual distance between two languages. Such a comparison can be helpful when there is a need to understand the differences between two programming languages, or also when trying to understand the differences between two modeling languages (or two variants of the same modeling language) that should be used for the

✉ Bernhard Rumpe
bernhard.rumpe@sosym.org

Jeff Gray
jeff.gray@sosym.org

[1] University of Alabama, Tuscaloosa, AL, USA

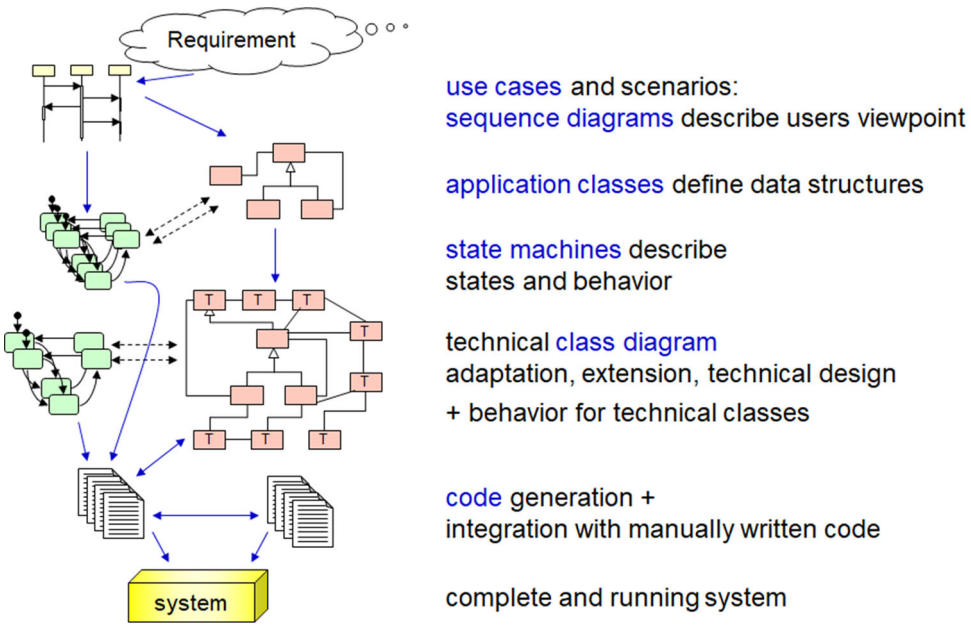
[2] RWTH Aachen University, Aachen, Germany

same purpose. Two possibilities exist for such comparisons: (1) When using meta-models (or grammars) to describe languages, we can apply conceptual distance techniques to the meta-models (or grammars) similar to the instance models mentioned earlier, and (2) we can also analyze the distance by mapping one language construct to a similar construct in the other language (we will call this encoding). This is very common when learning a new programming language by trying to understand how expressions, statements, case distinctions, loops, lambda-abstractions or functions are defined in terms of other known languages. This mapping also assists in identifying the absence of a familiar and helpful construct in one language that may not be available in another language. However, this approach does not always allow objective measurements, but only gives an informal understanding of the conceptual distance of languages.

Using the encoding approach, we might argue that as the conceptual distance between languages increases, the complexity of the encoding also increases.

conceptual distance still needs to be kept manageable. As a natural consequence, in software-intensive complex systems we can observe with increasing frequency that the development paradigm used in the early stages may influence the implementation. For example, using a component connector architecture for high-level models of automotive software will probably lead to the explicit existence of components, connectors and ports in the final implementation. The explicit existence of these constructs that exist in the modeling languages creates overhead in the implementation, but also keeps the development process manageable, more efficient and potentially less erroneous. Furthermore, variant management and reusability will probably increase. A similar phenomenon occurs in workflow-intensive systems, where workflow descriptions are based on modeling techniques to organize and understand the system. Such systems very explicitly know and manage the workflows and their respective states.

As a logical consequence, it can be observed that the form of the high-level modeling languages used to describe



These observations lead us to an interesting point about software engineering, from the simple viewpoint of starting with a set of informal requirements, which are mapped down through several high-level and low-level model formalizations (as shown in the figure), and finally end up in an executable implementation. Between each of those modeling levels, the conceptual distance must be small enough such that we can manage a systematic, sound mapping between the levels. Usually, these mappings are not exactly systematic encodings, but also add a lot of additional human-constructed design decisions. However, the a system varies much according to the shapes, structure and architecture of the implementation, as well as the abstractions available in the implementation. This helps to keep the conceptual distance between requirements and implementation small and the development manageable. Interestingly, this is different from other engineering domains (e.g., architecture), where the existence or form of models and plans has no impact on the created building.

There is much more to be said about conceptual distance. We look forward to seeing novel contributions to SoSyM

that describe investigations into either specific languages or as general considerations toward a theory of measuring conceptual distance.

1 Welcome to new editors!

With several editors retiring last year, along with an increase in submissions and new modeling topic areas emerging, we are happy to announce seven new distinguished members of the modeling community who have joined the SoSyM Editorial Board. They are: Silvia Abrahão, Nelly Bencomo, Didier Buchs, Dimitris Kolovos, Jan Mendling, Iris Reinhartz-Berger, and Davide Di Ruscio. We look forward to working with our new Editors in the future!

2 Content of this issue

This issue contains a Theme Section, two Special Sections and 14 Regular papers as follows:

1. Theme Section on model-based design of Cyber-Physical Systems
   Guest Editors: Manfred Broy, Heinrich Daembkes, and Janos Sztipanovits
2. MODELS 2016 Special Section
   Guest Editors: Jörg Kienzle and Alexander Pretschner
3. EMMSAD 2017 Special Section
   Guest Editors: Iris Reinhartz-Berger, Wided Guédria, and Palash Bera
4. Regular Papers
   - "A feature-based survey of model view approaches" by Hugo Bruneliere, Erik Burger, Jordi Cabot, and Manuel Wimmer
   - "Introducing probabilistic reasoning within Event-B" by Mohamed Aouadhi, Benoît Delahaye, and Arnaud Lanoix
   - "A model-driven framework for developing multi-agent systems in emergency response environments" by Samaneh HoseinDoost, Tahereh Adamzadeh, Bahman Zamani, and Afsaneh Fatemi
   - "An integrated metamodel-based approach to software model refactoring" by Mohammed Misbhauddin and Mohammad Alshayeb
   - "Template-based model generation" by Xiao He, Tian Zhang, Minxue Pan, Zhiyi Ma, and Chang-Jun Hu
   - "Transforming XML schemas into OWL ontologies using formal concept analysis" by Mokhtaria Hacherouf, Safia Nait-Bahloul, and Christophe Cruz
   - "Consolidation of database check constraints" by Nikola Obrenovic, Ivan Lukovic, and Sonja Ristic
   - "Multidimensional context modeling applied to non-functional analysis of software" by Luca Berardinelli, Marco Bernardo, Vittorio Cortellessa, and Antinisca Di Marco
   - "Enabling automated requirements reuse and configuration" by Yan Li, Tao Yue, Shaukat Ali, and Li Zhang
   - "Example-driven modeling: on effects of using examples on structural model comprehension, what makes them useful, and how to create them" by Dina Zayan, Atrisha Sarkar, Michal Antkiewicz, Rita Suzana Pitangueira Maciel, and Krzysztof Czarnecki
   - "Towards a model-driven engineering approach for the assessment of non-functional properties using multi-formalism" by Simona Bernardi, Stefano Marrone, Jose Merseguer, Roberto Nardone, and Valeria Vittorini
   - "Tradeoffs in modeling performance of highly configurable software systems" by Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, Alexander Grebhahn, and Sven Apel
   - "An integrated conceptual model for information system security risk management supported by enterprise architecture management" by Nicolas Mayer, Jocelyn Aubert, Eric Grandry, Christophe Feltus, Elio Goettelmann, and Roel Wieringa
   - "Execution of UML models: a systematic review of research and practice" by Federico Ciccozzi, Ivano Malavolta, and Bran Selic